合肥工業大學

操作系统实验报告

实验题目	实验 6 时间片轮转调度
学生姓名	袁焕发
学　　号	2019217769
专业班级	物联网工程 19-2
指导教师	田卫东
完成日期	2021 年 11 月 9 日

合肥工业大学 计算机与信息学院

1. 实验目的和任务要求

调试 EOS 的线程调度程序，熟悉基于优先级的抢先式调度。
为 EOS 添加时间片轮转调度，了解其它常用的调度算法。

2. 实验原理

EOS 为了实现基于优先级的抢先式调度，为线程定义了从 0 到 31 的 32 个优先级，其中 0 优先级最低，31 优先级最高。线程控制块结构体 THREAD（在文件 ps/psp.h 中定义）中的 Priority 域就是用来记录线程优先级的。线程调度最终由 PspSelectNextThread 函数决定是让被中断的线程继续执行，还是从所有“就绪”线程中选择一个来执行，也称函数 PspSelectNextThread 为“调度程序。

3. 实验内容

3.1. 准备实验

按 F7 生成在本实验创建的 EOS Kernel 项目，按 F5 启动调试，待 EOS 启动完毕，在 EOS 控制台中输入命令“rr”后按回车。

结束调试，在 ke/sysproc.c 文件的 ThreadFunction 函数中，调用 fprintf 函数的代码行（第 679 行）添加一个断点，启动调试。待 EOS 启动完毕，在 EOS 控制台中输入命令“rr”后按回车。“rr”命令开始执行后，会在断点处中断。刷新“进程线程”窗口，可以看到如图 所示的内容。其中，从 ID 为 24 到 ID 为 33 的线程是“rr”命令创建的 10 个优先级为 8 的线程，ID 为 24 的线程处于运行状态，其它的 9 个线程处于就绪状态。

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	16	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
7	24	Y	8	Running (2)	1	0x800188a2 ThreadFunction ← PspCurrentThread
8	25	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
9	26	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
10	27	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
11	28	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
12	29	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
13	30	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
14	31	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
15	32	Y	8	Ready (1)	1	0x800188a2 ThreadFunction
16	33	Y	8	Ready (1)	1	0x800188a2 ThreadFunction

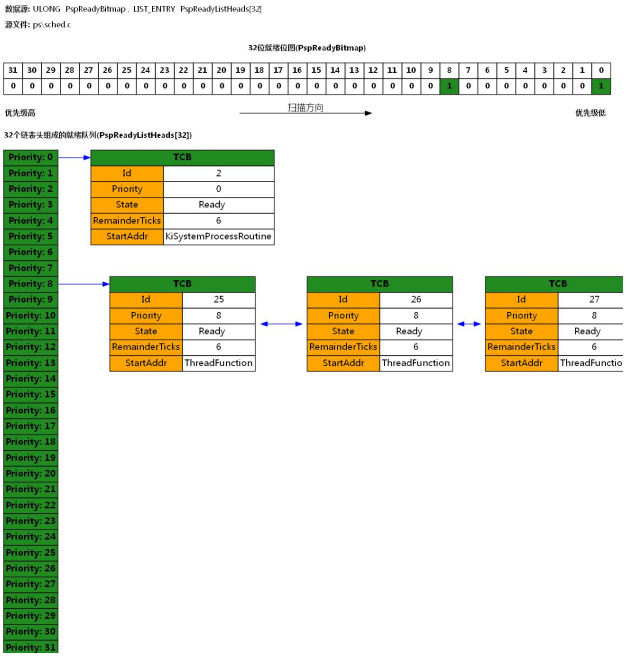
在“线程运行轨迹”窗口点击其工具栏上的“绘制指定范围线程”按钮，输入起始线程 ID 为 24 和结束线程 ID 为 33（需根据当前实际创建的线程 ID 进行调整），点击“绘制”按钮，可以查看这 10 个线程的运行状态和调度情况。

图例： 创建 就绪 运行 阻塞 结束

默认最多显示10个线程的120行数据，可使用本窗口工具栏上的“绘制指定范围的线程”按钮，查看需要显示的内容

Tick	TID=24	TID=25	TID=26	TID=27	TID=28	TID=29	TID=30	TID=31	TID=32	TID=33	函数	行号
512	创建										PspCreateThread	572
512	就绪										PspReadyThread	134
512											PspReadyThread	134
512		创建									PspCreateThread	572
512		就绪									PspReadyThread	134
512											PspReadyThread	134
512			创建								PspCreateThread	572
512			就绪								PspReadyThread	134
512											PspReadyThread	134
512				创建							PspCreateThread	572
512				就绪							PspReadyThread	134
512											PspReadyThread	134
512					创建						PspCreateThread	572
512					就绪						PspReadyThread	134
512											PspReadyThread	134
512						创建					PspCreateThread	572
512						就绪					PspReadyThread	134
512											PspReadyThread	134
512							创建				PspCreateThread	572
512							就绪				PspReadyThread	134
512											PspReadyThread	134
512								创建			PspCreateThread	572
512								就绪			PspReadyThread	134
512											PspReadyThread	134
512									创建		PspCreateThread	572
512									就绪		PspReadyThread	134
512											PspReadyThread	134
512										创建	PspCreateThread	572
512										就绪	PspReadyThread	134
512											PspReadyThread	134
513	运行										PspSelectNextThread	462
Tick	TID=24	TID=25	TID=26	TID=27	TID=28	TID=29	TID=30	TID=31	TID=32	TID=33	函数	行号

刷新“就绪线程队列”窗口，可以看到在就绪位图中，第 0 位和第 8 位为 1，其它位都为 0，相对应的，在就绪队列中只有优先级为 0 和优先级为 8 的就绪队列中挂接了处于就绪状态的线程，其它优先级的就绪队列都为空。其中，优先级为 0 的线程是空闲线程，9 个优先级为 8 的线程是从 ID 为 25 到 ID 为 33 的线程，而 ID 为 24 的线程此时正处于运行状态，所以不在优先级为 8 的就绪队列中。

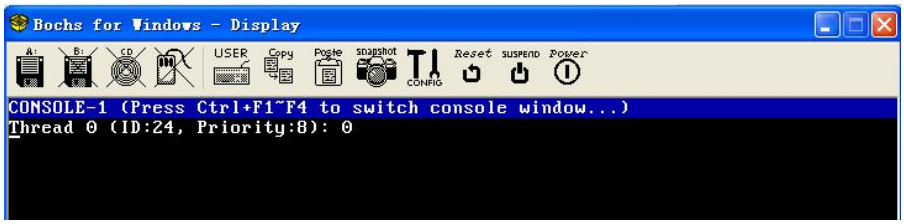


查看 ThreadFunction 函数中变量 pThreadParameter->Y 的值应该为 0，说明正在调试的是第 0 个新建的。

```
676         __asm("cli");
677         PsGetThreadPriority(CURRENT_THREAD_HANDLE, &Priority);
678         SetConsoleCursorPosition(pThreadParameter->StdHandle, CursorPosition);
679         fprintf(pThreadParameter->StdHandle, "Thread %d (ID:%d, Priority:%d): %u ",
680             pThreadParameter->Y, ObGetObjectId(PspCurrentThread),Priority, i);
681         __asm("sti");
682     }
```

pThreadParameter->Y = 0x0

激活虚拟机窗口，可以看到第 0 个新建的线程还没有在控制台中输出任何内容，原因是 fprintf 函数还没有执行。



3.2. 调试线程调度程序

3.2.1. 调试当前线程不被抢先的情况

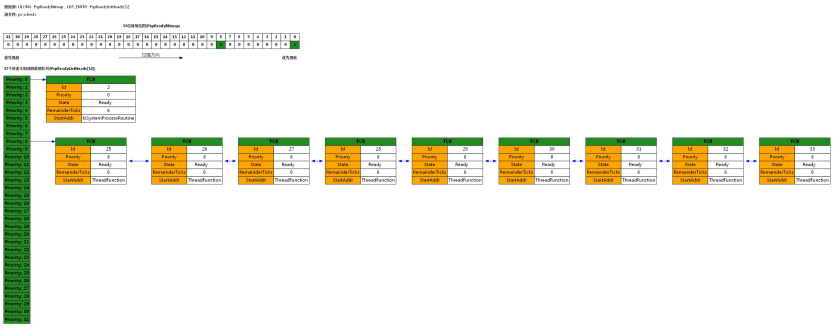
再次在“线程运行轨迹”窗口中点击工具栏上的“绘制指定范围线程”按钮，输入起始线程 ID 为 24 和结束线程 ID 为 33,点击“绘制”按钮，可以看到仍然是只有 ID 为 24 的线程处于运行状态，其它 9 个线程处于就绪状态。

图例： 创建 就绪 运行 阻塞 结束

默认最多显示10个线程的120行数据，可使用本窗口工具栏上的“绘制指定范围的线程”按钮，查看需要显示的内容

Tick	TID=24	TID=25	TID=26	TID=27	TID=28	TID=29	TID=30	TID=31	TID=32	TID=33	函数	行号
512	创建										PspCreateThread	572
512	就绪										PspReadyThread	134
512											PspReadyThread	134
512		创建									PspCreateThread	572
512		就绪									PspReadyThread	134
512											PspReadyThread	134
512			创建								PspCreateThread	572
512			就绪								PspReadyThread	134
512											PspReadyThread	134
512				创建							PspCreateThread	572
512				就绪							PspReadyThread	134
512											PspReadyThread	134
512					创建						PspCreateThread	572
512					就绪						PspReadyThread	134
512											PspReadyThread	134
512						创建					PspCreateThread	572
512						就绪					PspReadyThread	134
512											PspReadyThread	134
512							创建				PspCreateThread	572
512							就绪				PspReadyThread	134
512											PspReadyThread	134
512								创建			PspCreateThread	572
512								就绪			PspReadyThread	134
512											PspReadyThread	134
512									创建		PspCreateThread	572
512									就绪		PspReadyThread	134
512											PspReadyThread	134
512										创建	PspCreateThread	572
512										就绪	PspReadyThread	134
512											PspReadyThread	134
513	运行										PspSelectNextThread	462
Tick	TID=24	TID=25	TID=26	TID=27	TID=28	TID=29	TID=30	TID=31	TID=32	TID=33	函数	行号

继续之前的调试。在 ps/sched.c 文件的 PspSelectNextThread 函数中，调用 BitScanReverse 函数扫描就绪位图的代码行（第 391 行）添加一个断点。按 F5 继续执行，因为每当有定时计数器中断发生时（每 10ms 一次）都会触发线程调度函数 PspSelectNextThread，所以很快就会在刚刚添加的断点处中断。刷新“就绪线程队列”窗口，仍然会显示如下图所示的内容。



在“调试”菜单“窗口”中选择“监视”，在“监视”窗口中添加表达式“/t PspReadyBitmap”，以二进制格式查看就绪位图变量的值。此时就绪位图的值应该为 100000001，表示优先级为 8 和 0 的两个就绪队列中存在就绪线程。

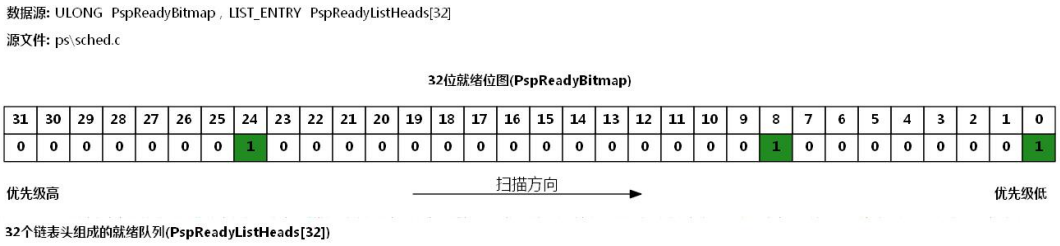
在“调试”菜单中选择“快速监视”，在“快速监视”对话框的“表达式”中输入表达式“*PspCurrentThread”后，点击“重新计算”按钮，可以查看当前正在运行的线程（即被中断的线程）的线程控制块中各个域的值。其中优先（Priority 域）的值为 8；状态（State 域）的值为 2（运行状态）；时间片（RemainderTicks 域）的值为 6；线程函数（StartAddr 域）为 ThreadFunction。综合这些信息即可确定当前正在运行的线程就是“rr”命令新建的第 0 个线程。

按 F10 单步调试，BitScanReverse 函数会从就绪位图中扫描最高优先级，并保存在变量 HighestPriority 中。查看变量 HighestPriority 的值为 8。

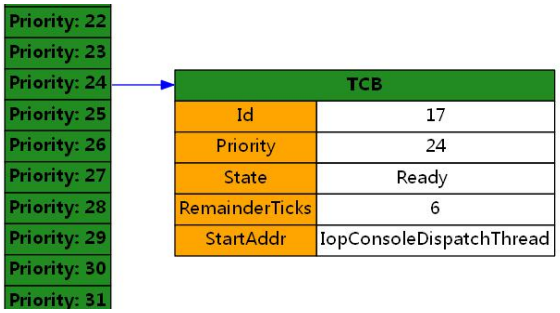
3.2.2. 调试当前线程被抢先的情况

删除之前添加的所有断点。在 ps/sched.c 文件的 PspSelectNextThread 函数的第 402 行添加一个断点。按 F5 继续执行，激活虚拟机窗口，可以看到第 0 个新建的线程正在执行。

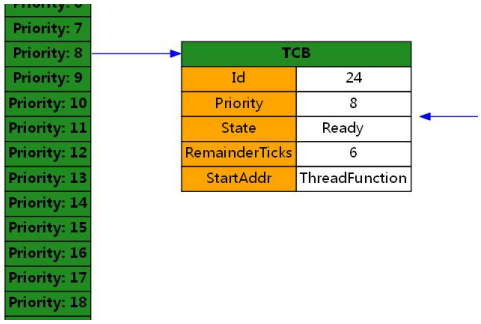
刷新“就绪线程队列”窗口，在 32 位就绪位图中第 24 位用绿色高亮显示且值为 1，说明优先级为 24 的就绪队列中存在就绪线程。



在“32 个链表头组成的就绪队列”中可以查看优先级为 24 的就绪队列中挂接了一个处于就绪状态的线程，如下图所示，通过其线程函数名称可以确认其为控制台派遣线程。



按 F10 单步调试到第 408 行。刷新“就绪线程队列”窗口，可以看到新建的第 0 个线程已经挂接在了优先级为 8 的就绪队列的队首，优先级为 8 的就绪队列中一共挂接了 10 个线程。



继续按 F10 单步调试，直到在 PspSelectNextThread 函数返回前（第 473

行）中断执行。此时，优先级为 24 的控制台派遣线程已经进入了运行状态，在中断返回后，就可以开始执行了。刷新“就绪线程队列”窗口，可以看到线程调度函数已经将控制台派遣线程移出了就绪队列。

数据源: ULONG PspReadyBitmap, LIST_ENTRY PspReadyListHeads[32]

源文件: ps\sched.c

32位就绪位图(PspReadyBitmap)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

优先级高 扫描方向 优先级低

刷新“进程线程列表”窗口，可以看到当前线程指针 PspCurrentThread 已经指向了控制台派遣线程。

线程列表

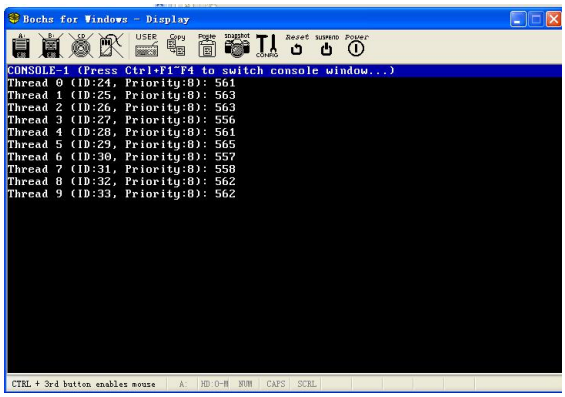
序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Running (2)	1	0x80015724 IopConsoleDispatchThread ← PspCurrentThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread

3.3. 为 EOS 添加时间片轮转调度算法

3.3.1. 要求及测试方法

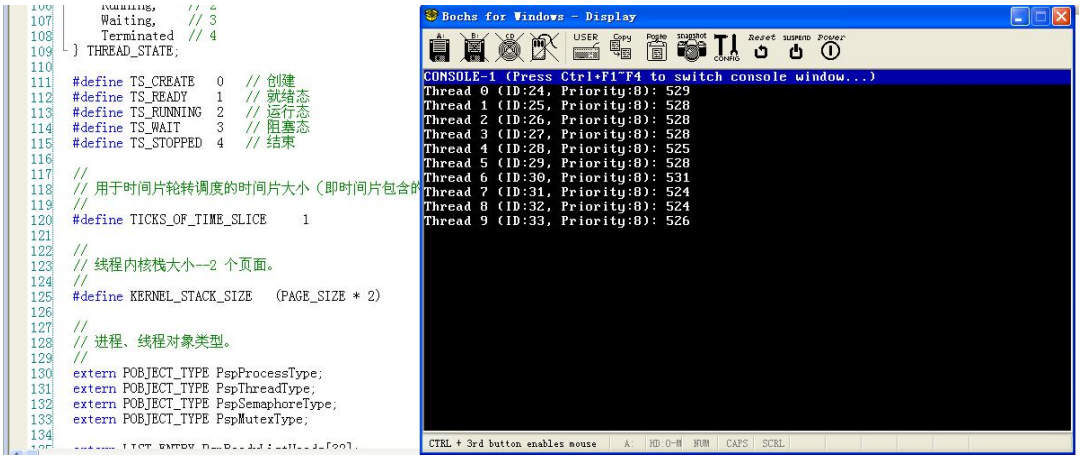
修改 ps/sched.c 文件中的 PspRoundRobin 函数(第 344 行)，在其中实现时间片轮转调度算法。代码修改完毕后，生成 EOS 内核项目，启动调试。在 EOS 控制台中输入命令“rr”后按回车。应能看到 10 个线程轮转执行的效果，如下图所示。

```
if((PspCurrentThread!=NULL)&&(Running==PspCurrentThread->State)){
    PspCurrentThread->RemainderTicks--;
    if(PspCurrentThread->RemainderTicks==0)
    {
    PspCurrentThread->RemainderTicks=TICKS_OF_TIME_SLICE;
    }
    if(BIT_TEST(PspReadyBitmap,PspCurrentThread->Priority)){
    PspReadyThread(PspCurrentThread);
    }
    }
    return;
```

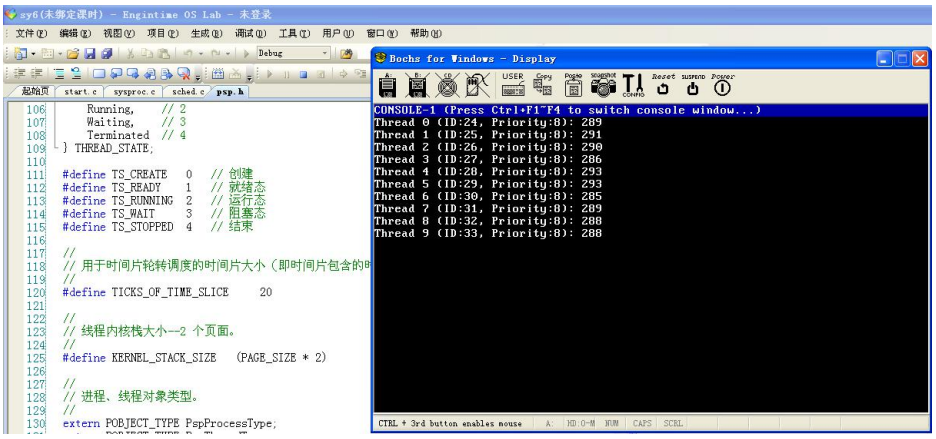


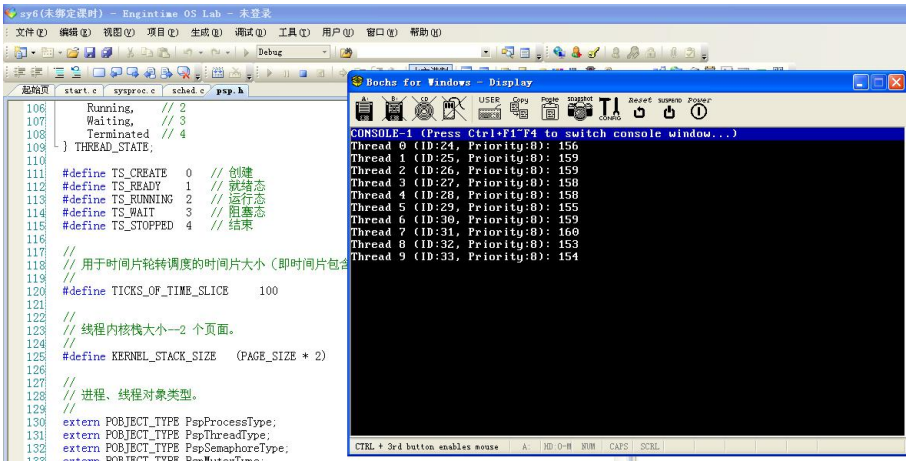
3.4. 修改线程时间片的大小

将 ps/psp.h 第 120 行定义的 TICKS_OF_TIME_SLICE 的值修改为 1，生成 EOS 内核项目，启动调试，在 EOS 控制台中输入命令“rr”后按回车，观察执行的效果。



还可以按照上面的步骤为 TICKS_OF_TIME_SLICE 取一些其它的极端值，例如 20 或 100 等，分别观察“rr”命令执行的效果。





4. 实验的思考与问题分析

1. 结合线程调度执行的时机，说明在 ThreadFunction 函数中，为什么可以使用“关中断”和“开中断”的方法来保护控制台这种临界资源。一般情况下，应该使用互斥信号量（MUTEX）来保护临界资源，但是在 ThreadFunction 函数中却不能使用互斥信号量，而只能使用“关中断”和“开中断”的方法，结合线程调度的对象说明这样做的原因。

答:

如果使用互斥信号量，由于访问临界区而被阻塞的线程，会被放入互斥信号量的等待队列，会改变这些线程的状态。开中断和关中断使处理机在这段时间屏蔽掉了外界所有中断，其他线程无法占用资源。使用开中断和关中断进程同步不会改变线程状态，可以保证那些没有获得处理器的资源都在就绪队列中。关中断后 CPU 就不会响应任何由外部设备发出的硬中断，也就不会发生线程调度了，从而保证各个线程可以互斥的访问控制台。

2. 时间片轮转调度发现被中断线程的时间片用完后，而且在就绪队列中没有与被中断线程优先级相同的就绪线程时，为什么不需要将被中断线程转入“就绪”状态？如果此时将被中断线程转入了“就绪”状态又会怎么样？可以结合 PspRoundRobin 函数和 PspSelectNextThread 函数的流程进行思考，并使用抢先和不抢先两种情况进行说明。

答:

通过监视 PspRoundRobin 函数和 PspSelectNextThread 函数的流程，当时间片轮转调度发现被中断线程的时间片用完后，而且在就绪队列中没有与被中断线程优先级相同的就绪线程时，PspRoundRobin 函数会直接结束，所以不需要将被中断线程转入“就绪”状态。如果此时将被中断线程转入了“就绪”状态，那么比该中断线程更高的就绪进程就无法运行。

3. 在 EOS 只实现了基于优先级的抢先式调度时，同优先级的线程只能有一个被执行。当实现了时间片轮转调度算法后，同优先级的线程就能够轮流执行从而获得均等的执行机会。但是，如果有高优先级的线程一直占用 CPU，低优先级的线程就永远不会被执行。尝试修改 ke/sysproc.c 文件中的 ConsoleCmdRoundRobin 函数来演示这种情况（例如在 10 个优先级为 8 的线程执行时，创建一个优先级为 9 的线程）。设计一种调度算法来解决此问题，让低优先级的线程也能获得被执行的机会。

答：

实现动态优先级算法。动态优先级是指在创建进程是所赋予的优先级，可随线程的推进而改变，以便获得良好的性能调度。例如，在就绪队列中的线程，随着其等待时间的增长，其优先级以速率 x 增加，并且正在执行的线程，其优先级以速率 y 下降。在各个线程具有不同优先级的情况下，对于优先级低的线程，在等待足够的时间后，其优先级便可能升为最高，从而获得被执行的机会。此时，在基于优先级的抢占式调度算法、时间片轮转调度算法和动态优先级算法的共同作用下，可防止一个高优先级的长作业长期的垄断处理器。

4. EOS 内核时间片大小取 60ms（和 Windows 操作系统完全相同），在线程比较多时，就可以观察出线程轮流执行的情况（因为此时一次轮转需要 60ms，10 个线程轮流执行一次需要 60×10＝600ms，所以 EOS 的控制台上可以清楚地观察到线程轮流执行的情况）。但是在 Windows、Linux 等操作系统启动后，正常情况下都有上百个线程在并发执行，为什么觉察不到它们被轮流执行，并且每个程序都运行的很顺利呢？

答：

两者虽然时间片大小一致，但是在 Windows 操作系统中，运行几百个线程也不会将 CPU 全部占用，CPU 的整体利用率很低，而且 CPU 处理一个线程往往不需要一个时间片就能完成，在执行一个线程时，其他线程大多都处于阻塞状态，阻塞的原因主要是等待 I/O 完成或者等待命令消息的到达，也就是等待用户进行交互，操作系统才根据用户需求进行相关程序的处理。

5. 总结和感想体会

通过这次实验，明白了基于优先级的抢先式调度算法的运行流程。当这种线程调度方式运行时，如有比正在执行的线程优先级高的线程处于“就绪”状态，这种调度方式会停止正在执行的低优先级的线程，然后将处理器分配给高优先级的线程，使之执行，而低优先级的线程会进入“就绪”状态，直到再也没有比它优先级高的“就绪”线程时，它才能重新获得处理器。理解了时间片轮转调度要为就绪队列中的每个线程分配一个时间片，当线程调度执行时，把 CPU 分配给队首线程，待线程的时间片用完后，会重新为它分配一个时间片，并将它移动到就绪队列的末尾，从而让新的队首线程开始执行，最终在指导下修改 PspRoundRobin 函数实现时间片轮转调度。